Como criar seu primeiro aplicativo Web usando Flask e Python

1. Introdução

O Flask é um microframework Python para desenvolvimento ágil de aplicativos web, adequado tanto para iniciantes quanto para desenvolvedores mais experientes. Ele é bastante leve e extensível, permitindo expandir facilmente seu aplicativo para operar com bibliotecas mais avançadas, aproveitando todo o poder da linguagem Python e a flexibilidade da web. Neste tutorial, você criará um aplicativo que exibe texto HTML, aprenderá sobre roteamento de aplicativos web, interação através de rotas de conteúdo estático e dinâmico, além de utilizar o depurador para corrigir eventuais erros.

2. Implantação do Ambiente

Pré-Requisitos

Siga as instruções inicias contidas no repositório IHCEUB para implantação do ambiente de laboratório, certificando-se de ter compreendido a implantação da VM com Docker que atuará como servidor web, além das ferramentas de gerenciamento, incluindo acesso remoto via SSH e editor de textos Vim, cujos fundamentos e comandos essenciais foram introduzidos em sala de aula.

Criando o aplicativo

Acesse o ambiente via SSH e vá até o diretório `/opt/ihceub/flask`. Crie o arquivo `app.py` com o Vim (use o comando `vim app.py`) ou editor de sua preferência. Insira o código abaixo para o seu primeiro aplicativo Flask com a implementação do "Hello World!":

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello():
    return "Olá, mundo!"

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

Salve o arquivo. Se estiver no Vim, lembre-se de usar o comando `:wq!`.

Executando o aplicativo

Vá até o diretório `/opt/ihceub/flask` e suba o contêiner do Flask.

```
docker-compose build
docker-compose up -d
```

Verifique se o contêiner está ativo e sem erros de implantação.

```
docker-compose ps
docker-compose logs
```

Agora, acesse `http://127.0.0.1:8500/` em seu navegador e você verá "Olá, mundo!".

3. Roteamento e visualizações

Roteamento refere-se ao mapeamento de URLs específicas para funções em um aplicativo web. Em outras palavras, quando você acessa um determinado endereço em um navegador web (ou através de uma chamada API), o aplicativo precisa saber qual função deve ser executada e o que deve ser retornado para o usuário. No Flask, isso é feito através do uso de decoradores, como `@app.route()`, para associar funções específicas a URLs. Por exemplo:

```
@app.route('/inicio')
def inicio():
    return "Página Inicial"
```

Dito isso, vamos adicionar mais algumas rotas ao nosso aplicativo. Vá até o diretório `/opt/ihceub/flask`, edite o arquivo `app.py` e acrecente:

```
@app.route('/sobre')
def sobre():
    return "Sobre o aplicativo..."

@app.route('/contato')
def contato():
    return "Página de contato."
```

Agora reinicialize o contêiner:

```
docker-compose down && docker-compose up -d
```

Verifique se o contêiner está ativo e sem erros de implantação.

```
docker-compose ps
docker-compose logs
```

Dessa forma, você poderá acessar os *end-points* `http://127.0.0.1:8500/sobre` ou `http://127.0.0.1:8500/contato`, e verá as respectivas páginas em seu navegador.

Nota 1: A identação do código é fundamental em Python pois, ao contrário de outras linguagens de programação que usam `{}` ou palavras-chave específicas para delimitar blocos de instruções, Python utiliza a identação para este propósito. Assim, a identação passa a determinar como os comandos são agrupados e, consequentemente, a estrutura e a lógica do seu programa. Por exemplo, vamos considerar um simples código com uma estrutura condicional:

```
#Código Correto (com identação apropriada):
if 5 > 3:
    print("5 é maior que 3")
else:
    print("5 não é maior que 3")
```

No caso acima, "5 é maior que 3" será impresso porque a expressão é verdadeira. Assim, a instrução print dentro do bloco if será executada.

```
#Código incorreto (sem identação apropriada):
if 5 > 3:
print("5 é maior que 3")
else:
print("5 não é maior que 3")
```

Já o código acima produzirá um erro de sintaxe porque Python espera que o bloco sob o if esteja identado. A identação não é apenas estética. Ela define a estrutura do código e determina quais instruções pertencem a qual bloco. Se duas instruções estiverem identadas no mesmo nível, elas pertencem ao mesmo bloco.

Nota 2: Lembre-se que o trecho de código indicado abaixo deve ser mantido por último, pois é a parte do programa responsável por executar o Flask quando o Python invocar o script `app.py` por meio do Docker.

```
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

4. Rotas Dinâmicas

Vamos permitir que os usuários interajam com o aplicativo por meio de rotas dinâmicas. Adicione o seguinte trecho ao seu código:

```
@app.route('/usuario/<nome>')
def saudacao(nome):
    return f"Olá, {nome}!"
```

Salve o script `app.py` e reinicialize o contêiner:

```
docker-compose down && docker-compose up -d
```

Verifique se o contêiner está ativo e sem erros de implantação:

```
docker-compose ps
docker-compose logs
```

Pronto, se você acessar `http://127.0.0.1:8500/usuario/Jose`, verá "Olá, Jose!" em seu navegador.

5. Depurando seu aplicativo

O Flask possui um depurador embutido. No nosso ambiente, quando você executa o comando `docker-compose logs`, poderá verificar quais são os eventuais erros e assim corrigir o código de seu aplicativo. Vamos imaginar que você tenha o seguinte erro de sintaxe em seu código Python:

```
#Aqui, você esqueceu de fechar o parêntese em uma chamada de função:
print("Olá, mundo!"
```

Se você tentar executar este código a partir de um contêiner, a saída do comando `docker-compose logs` deverá retornar um erro com este padrão:

```
web_1  | Traceback (most recent call last):
web_1  |   File "app.py", line 1, in <module>
web_1  |     print("Olá, mundo!"
web_1  |                        ^
web_1  | SyntaxError: unexpected EOF while parsing
```

Ou seja, uma página de erro detalhada será exibida nos logs, ajudando a identificar o problema. Como já ativamos o ambiente de desenvolvimento, o depurador está habilitado por padrão. Em ambientes de produção, você pode utilizar um web server mais robusto como o Gunicorn e uWSGI (veremos isso nas próximas aulas). Neste caso, para ativar o modo de depuração, você poderia acrescentar a diretiva `debug=true` ao aplicativo.

```
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=true)
```

Conclusão

Vamos entender o código parte por parte:

```python
#Aqui, estamos importando a classe Flask da biblioteca flask.
from flask import Flask

#Aqui estamos criando uma instância da classe Flask e atribuindo-a à
variável app. O argumento __name__ é uma variável especial
#que retorna o nome do módulo atual. Em scripts executados diretamente,
__name__ é igual a __main__. Isso indica ao Flask onde
#começar a procurar por coisas como templates e arquivos estáticos.
app = Flask(__name__)

#Estamos definindo uma rota para o endereço base ('/') da nossa aplicação.
Quando o usuário acessar o endereço base, a função
#hello() será chamada e retornará a string "Olá, mundo!".
@app.route('/')
def hello():
    return "Olá, mundo!"

#Aqui, definimos outra rota ('/sobre'). Quando o usuário acessar essa
rota, ele verá a mensagem "Sobre o aplicativo...".
@app.route('/sobre')
def sobre():
    return "Sobre o aplicativo..."

#Similar ao anterior, essa rota direciona o usuário para a página de
contato, retornando a mensagem "Página de contato.".
@app.route('/contato')
def contato():
    return "Página de contato."

#Esta rota é um pouco mais avançada. Ela espera um valor dinâmico na URL.
Por exemplo, se você acessar '/usuario/Joao',
#a palavra 'Joao' será capturada e passada como argumento para a função
saudacao(). O resultado será "Olá, Joao!".
@app.route('/usuario/<nome>')
def saudacao(nome):
    return f"Olá, {nome}!"

#Essa é a parte do código que executa a aplicação. O código dentro do
bloco if só será executado se o script for rodado
#diretamente (e não importado em outro script). No caso, estamos fazendo a
chamada do script com Docker, conforme definido
#em nosso Dockerfile e docker-compose.yml
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

Nota 1: O trecho `host='0.0.0.0'` indica que o servidor ficará acessível para qualquer IP que tenha conectividade ao servidor Flask. Usar `0.0.0.0` em ambientes de produção pode expor o aplicativo a riscos desnecessários. Dessa forma, em produção, costumamos indicar explicitamente o endereço IP autorizado a invocar o aplicativo, geralmente um proxy reverso. Detalharemos esta arquitetura em nossas próximas aulas.

Nota 2: O trecho `port=5000` define a porta onde o servidor estará rodando. O padrão do Flask é 5000. Mas em nossa VM de laboratório, exploramos o conceito de NAT e estamos utilizando a porta 8500. Então, ao executar esse script e acessar `http://127.0.0.1:8500` no seu navegador, você verá a mensagem "Olá, mundo!". Ao acessar `http://127.0.0.1:8500/sobre`, você verá "Sobre o aplicativo...", e assim por diante.

Pronto!

Você criou um pequeno aplicativo web com o Flask, adicionou rotas estáticas e dinâmicas e aprendeu a usar o depurador. A partir daqui, você pode expandir seu aplicativo, integrando-o com bancos de dados, formulários e aprimorando seu visual com CSS e HTML. Nos próximos laboratórios veremos como aplicar estes recursos em maiores detalhes.